Random Separation: A New Method for Solving Fixed-Cardinality Optimization Problems

Leizhen Cai*, Siu Man Chan, and Siu On Chan

Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, Hong Kong SAR, China
{lcai, smchan2, sochan2}@cse.cuhk.edu.hk

Abstract. We develop a new randomized method, *random separation*, for solving fixed-cardinality optimization problems on graphs, i.e., problems concerning solutions with exactly a fixed number k of elements (e.g., k vertices V') that optimize solution values (e.g., the number of edges covered by V'). The key idea of the method is to partition the vertex set of a graph randomly into two disjoint sets to separate a solution from the rest of the graph into connected components, and then select appropriate components to form a solution. We can use universal sets to derandomize algorithms obtained from this method.

This new method is versatile and powerful as it can be used to solve a wide range of fixed-cardinality optimization problems for degree-bounded graphs, graphs of bounded degeneracy (a large family of graphs that contains degree-bounded graphs, planar graphs, graphs of bounded tree-width, and nontrivial minor-closed families of graphs), and even general graphs.

1 Introduction

1.1 Motivations and Related Work

Many NP-hard problems, when some part of input I is taken as a fixed parameter k to form *fixed-parameter problems*, can be solved by algorithms that run in *uniformly polynomial time*, i.e., $f(k)|I|^{O(1)}$ time for some function $f(k)$. Prominent and influential examples of such algorithms include $O(n^3)$ algorithms of Robertson and Seymour [12] for solving the subgraph homomorphism and minor containment problems, an $O(n)$ algorithm of Bodlaender [3] for finding tree-decompositions of tree-width k, $O(n)$ algorithms of Courcelle [7] and Arnborg *et al.* [2] for solving problems expressible in monadic second-order logic on graphs of tree-width k, and an $O(kn + 1.286^k)$ algorithm of Chen, Kanj and Jia [6] for finding a vertex cover of k vertices. Downey and Fellows [8] have established a general framework for studying the complexity of fixed-parameter problems.

In this paper, we develop a new randomized method, called *random separation*, for designing uniformly polynomial-time algorithms to solve fixed-parameter

* Partially supported by a Direct Research Grant from the Chinese University of Hong Kong.

problems on graphs, especially fixed-cardinality optimization problems. A *fixed-cardinality optimization problem* is a problem that asks for a solution with exactly a fixed number k of elements (e.g., k vertices V') to optimize the solution value (e.g., the number of edges covered by V'), and recently Cai [4] has initiated a systematic study of fixed-cardinality optimization problems from the parameterized complexity point of view. The key idea in our random separation method is to partition the vertex set of a graph randomly into two disjoint sets to separate a solution from the rest of the graph into connected components, and then select appropriate components to form a solution. We can use universal sets to derandomize algorithms obtained from this method.

Our initial inspiration for developing the random separation method came from the colour-coding method of Alon, Yuster and Zwick [1] for solving fixed-parameter problems on graphs. The basic idea of their method is to colour vertices randomly in k colours and then try to find a *colourful k-solution*, i.e., a solution consisting of k vertices in distinct colours. To derandomize the algorithm, they use perfect hash functions. They have used colour-coding to find, for each fixed k, a k-path in $O(n)$ expected time and $O(n \log n)$ worst-case time, a k-cycle in $O(n^\alpha)$ expected time and $O(n^\alpha \log n)$ worst-cast time, where $\alpha < 2.376$, and a subgraph isomorphic to a k-vertex graph H of tree-width w in $O(n^{w+1})$ expected time and $O(n^{w+1} \log n)$ worst-case time. Unfortunately, for most fixed-parameter problems, it seems very difficult to find colourful k-solutions, which greatly limits the applicability of colour-coding.

On the other hand, it is much easier to deal with connected components, which enables us to use random separation to solve a wide range of fixed-parameter problems, especially when the input graph has bounded degree or degeneracy. In fact, it is rather surprising that we can use random separation to obtain uniformly polynomial-time algorithms for classes of fixed-parameter problems, especially fixed-cardinality optimization problems, on graphs of bounded degree.

For derandomization, our main tools are universal sets and perfect hash functions. A collection of binary vectors of length n is *(n, t)-universal* if for every subset of size t of the indices, all 2^t configurations appear. Naor, Schulman and Srinivasan [11] have a deterministic construction for (n, t)-universal sets of size $2^t t^{O(\log t)} \log n$ that can be listed in linear time. A family $\mathcal{F}$ of functions mapping a domain of size n into a range of size k is an *(n, k)-family of perfect hash functions* if for every subset S of size k from the domain there is a function in $\mathcal{F}$ that is 1-to-1 on S. Based on work of Schmidt and Siegel [13] and pointed out by Moni Naor [1], an (n, k)-family of perfect hash functions of size $2^{O(k)} \log n$ can be deterministically constructed in linear time.

Following the framework of Downey and Fellows [8], we say that a fixed-parameter problem is *fixed-parameter tractable* if it has a uniformly polynomial-time algorithm, and *fixed-parameter intractable* if it is W[i]-hard for some W[i] in the W-hierarchy. We note that a W[i]-hard problem cannot be solved in uniformly polynomial time unless all problems in W[i] can be solved in uniformly polynomial time.

1.2 Main Results

We focus on graphs of bounded degree or degeneracy as many fixed-parameter problems are fixed-parameter intractable for general graphs. A *degree-bounded graph* is a graph whose maximum degree is bounded by a constant d. A graph G is *d-degenerate* if every induced subgraph of G has a vertex of degree at most d. It is easy to see that every d-degenerate graph admits an acyclic orientation such that the outdegree of each vertex is at most d. Many interesting families of graphs are d-degenerate for some fixed constant d. For example, graphs embeddable on some fixed surface (planar graphs are 5-degenerate), degree-bounded graphs, graphs of bounded tree-width, and nontrivial minor-closed families of graphs.

In this paper we will demonstrate the power of our random separation method by the following results:

1. For degree bounded graphs G, we obtain uniformly polynomial-time algorithms for a wide range of fixed-parameter problems that ask us to find k vertices (edges) S to optimize a value $\phi(S)$ defined by an objective function ϕ that is computable in uniformly polynomial time (Section 3.2).
2. For every degree bounded graph G and every k-vertex graph H, we can find a maximum (minimum) weight induced (partial) H-subgraph in G in $O(n \log n)$ time for each fixed k (Section 2.2).
3. For each fixed k, we can find a subset of vertices in a general graph to cover exactly k edges in $O(m + n \log n)$ time (Section 2.4).
4. For every k-vertex tree (forest) H and fixed k and d, we can find an induced H-subgraph in a d-degenerate graph that contains one in $O(n)$ expected time and $O(n \log^2 n)$ worst-case time (Section 4).
5. For fixed k and d, we can find an induced k-cycle in a d-degenerate graph that contains one in $O(n^2)$ expected time and $O(n^2 \log^2 n)$ worst-case time (Section 4).

Furthermore, we can also use random separation to solve fixed-parameter problems on satisfiability, integer programming, set packing and covering, and many others when the input obeys some "degree" constraints, which will be discussed in the full paper.

1.3 Notation and Organization

We use $G = (V, E)$ to denote the input graph (or digraph) with n vertices and m edges. For a graph H, its vertex (edge) set is denoted by $V(H)$ (respectively, $E(H)$). For a subset V' of vertices, $N_G(V')$ denotes the *neighbourhood* of V', i.e., the set of vertices not in V' that are adjacent to some vertices in V', and $N_G^+(V')$ the *out-neighbourhood* of V', i.e., vertices not in V' that are heads of edges connected with some vertices in V'. For a subgraph H, we use $N_G(H)$ as a shorthand for $N_G(V(H))$ and $N_G^+(H)$ for $N_G^+(V(H))$. We use $d_G(v)$ to denote the degree of vertex v in G.

A subgraph in G that is isomorphic to a given graph H is an *H-subgraph*. For two vertices u and v, $d_G(u, v)$ denotes their distance in G; and for two subgraphs

H_1 and H_2 of G, $d_G(H_1, H_2)$ denotes their distance in G, i.e., $d_G(H_1, H_2) = \min\{d_G(u, v) \mid u \in V(H_1), v \in V(H_2)\}$.

In Section 2, we introduce our basic random separation method together with several working examples. In Section 3, we extend the method to solve a wide range of fixed-cardinality optimization problems for degree-bounded graphs, and in Section 4 we combine random separation with colour-coding to find induced k-vertex trees (forests) and induced k-cycles for d-degenerate graphs. We conclude the paper with a brief summary and some open problems in Section 5.

2 Random Separation

The basic idea of our random separation method is to use a random partition of the vertex set V of a graph $G = (V, E)$ to separate a solution from the rest of G into connected components and then select appropriate components to form a solution. To be precise, we colour each vertex randomly and independently by either green or red to define a random partition of V into green vertices V_g and red vertices V_r, which forms the *green subgraph* $G_g = G[V_g]$ and the *red subgraph* $G_r = G[V_r]$. For a solution S with k vertices, there is $2^{-(k+|N_G(S)|)}$ chance that a random partition has the property that S is entirely in the green subgraph G_g and its neighbourhood $N_G(S)$ is entirely in the red subgraph G_r, i.e., S consists of a collection of connected components of G_g, referred to as *green components.* For such a partition, we can usually find an appropriate collection of green components to form a required k-solution by using the standard dynamic programming algorithm for the 0-1 knapsack problem (see, for instance, the textbook of Kleinberg and Tardos [10]). Therefore, with probability $2^{-(k+|N_G(S)|)}$, we can find a required k-solution from a random partition. To derandomize the algorithm, we use (n, t)-universal sets for $t = k + |N_G(S)|$. The total time of the whole algorithm is uniformly polynomial when t is bounded by a function of k. We give several examples to illustrate this method in the rest of this section.

2.1 Dense k-Vertex Subgraphs in Degree-Bounded Graphs

Let us start with the problem of finding an induced subgraph on k vertices that contains the maximum number of edges. Let d be a fixed constant and $G = (V, E)$ a graph of maximum degree d. First we randomly colour each vertex of G by either green or red to form a random partition (V_g, V_r) of V. Let G' be a maximum k-vertex induced subgraph of G. A partition of V is a "good partition" for G' if all vertices in G' are green and all vertices in its neighbourhood $N_G(G')$ are red. Note that $N_G(G')$ has at most dk vertices as $d_G(v) \le d$ for each vertex v. Therefore the probability that a random partition is a good partition for G' is at least $2^{-(d+1)k}$ and thus, with at least such a probability, G' is the union of some green components.

To find a maximum k-vertex induced subgraph for a good partition of G', we need only find a collection $\mathcal{H}'$ of green components such that the total number of vertices in $\mathcal{H}'$ is k and the total number of edges in $\mathcal{H}'$ is maximized. For this purpose, we first compute in $O(dn)$ time the number n_i of vertices and the number m_i of edges inside each green component H_i. Then we find a collection $\mathcal{H}'$ of green components that maximizes

$$\sum_{H_i \in \mathcal{H}'} m_i$$

subject to $\sum_{H_i \in \mathcal{H}'} n_i = k$. This can be solved in $O(kn)$ time by using the standard dynamic programming algorithm for the 0-1 knapsack problem (see [10]). Therefore, with probability at least $2^{-(d+1)k}$, we can find a maximum k-vertex induced subgraph of G in $O((d+k)n)$ time.

To derandomize the algorithm, we need a family of partitions such that for every partition Π of any $(d+1)k$ vertices into k vertices and dk vertices, there is a partition in the family that is consistent with Π. Clearly, any family of $(n,(d+1)k)$-universal sets can be used as the required family of partitions. Using a construction of Naor, Schulman and Srinivasan [11], we obtain a required family of partitions of size $2^{(d+1)k}(dk+k)^{O(\log(dk+k))}\log n$ that can be listed in linear time. Therefore we obtain a deterministic algorithm that runs in $O(f(k,d)n\log n)$ time where

$$f(k,d) = 2^{(d+1)k}(dk+k)^{O(\log(dk+k))}(d+k),$$

which is $O(n\log n)$ for fixed k and d, and uniformly polynomial for parameter k and fixed d.

2.2 Subgraph Isomorphism for Degree-Bounded Graphs

Although subgraph isomorphism problems are W[1]-hard for general graphs [8], we can use random separation to solve them easily for degree-bounded graphs, even for the weighted case. We note that the following theorem for the unweighted case also follows from a result of Seese [14] and a more general result of Frick and Grohe [9].

Theorem 1. *Let $G = (V, E; w)$, where $w : V \bigcup E \to R$, be a weighted graph (digraph) whose maximum degree is d, and H an arbitrary k-vertex graph (digraph). If G contains an induced (a partial) H-subgraph, then for fixed k and d, it takes $O(n\log n)$ time to find a maximum (minimum) weight induced (partial) H-subgraph in G.*

Proof. We consider induced subgraph first. Let H' be a maximum (minimum) weight induced H-subgraph in G. Generate a random partition (V_g, V_r) of V. With probability at least $2^{-(d+1)k}$, each connected component of H' is a green component. For each connected component H_i of H, we find a maximum (minimum) weight H_i-subgraph H_i^* in G_g, which takes $O(k!kdn)$ time. Then with probability at least $2^{-(d+1)k}$, $\cup_{H_i \in H} H_i^*$ is a maximum (minimum) weight induced H-subgraph in G, and therefore we can solve the problem in $O(n)$ expected time for fixed k and d. We derandomize the algorithm by using $(n,(d+1)k)$-universal sets to obtain a deterministic algorithm that runs in $O(n\log n)$ time for fixed k and d.

For the partial subgraph case, we use a random partition to partition edges E into green and red edges (E_g, E_r). Let H' be a maximum (minimum) weight partial H-subgraph in G. With probability at least 2^{-kd} (note that k vertices are incident with at most kd edges), edges in H' are green and all other edges adjacent to edges in H' are red, i.e., each connected component of H' is a green component in $G[E_g]$. The rest of the arguments is the same as the induced subgraph case and is omitted. $\square$

2.3 Weighted Independent *k*-Sets in *d*-Degenerate Graphs

Let d be a fixed constant, and $G = (V, E; w)$ a weighted d-degenerate graph with $w : V \to R$. Consider the problem of finding a maximum weight independent k-set in G, i.e., a set of k mutually nonadjacent vertices of maximum total weight. Although the problem is W[1]-hard for general graphs [8], it is trivially solvable for d-degenerate graphs by using random separation.

First we orient edges of G so that the outdegree of each vertex is at most d, which is easily done in $O(dn)$ time. Then we generate a random partition (V_g, V_r) of V. The probability that a maximum weight independent k-set V' is entirely inside G_g and the out-neighbourhood of each vertex of V' is entirely in G_r is at least $2^{-(d+1)k}$. Therefore, with probability at least $2^{-(d+1)k}$, V' consists of sinks of G_g and thus k sinks of largest weights in G_g. We can easily find such a V' in $O(kn)$ time, and thus, with probability at least $2^{-(d+1)k}$, we can find a maximum weight independent k-set in $O((d + k)n)$ time. Again, we can derandomize the algorithm by using $(n, (d+1)k)$-universal sets to obtain a deterministic algorithm that runs in $O(n \log n)$ time for fixed k and d.

Remark 2. For fixed constants k and d, we can also use random separation to find a maximum weight induced k-matching in d-degenerate graphs in $O(n \log n)$ time, and in Section 4 we will combine with colour coding to solve several other induced subgraph isomorphism problems for d-degenerate graphs.

2.4 Covering Exactly *k* Edges in General Graphs

We now consider the problem of finding a set of vertices to cover exactly k edges in a general graph G. W.l.o.g., we may assume that G has no isolated vertices. Let V' be a set of vertices that cover exactly k edges. Clearly $|V'| \leq k$ and thus every vertex in V' has degree at most k. Let V_k be the set of vertices of degree at most k in G and $F = G[V_k]$.

A random partition of V_k is a "good partition" for V' if all vertices in V' are green and all vertices in $N_F(V')$ are red. Since $N_F(V')$ has at most k vertices, the probability that a random partition is a good partition for V' is at least 2^{-2k}. Given a good partition for V', the problem of finding a subset of vertices to cover exactly k edges is equivalent to the problem of finding a collection $\mathcal{H}'$ of green components such that the total number of edges in G covered by vertices in $\mathcal{H}'$ is exactly k. To find such an $\mathcal{H}'$, we compute, for each green component H_i, the number e_i of edges in G covered by vertices in H_i. Since for any two green components H_i and H_j, the number of edges covered by vertices in $H_i \cup H_j$ equals $e_i + e_j$, we can obtain such a collection $\mathcal{H}'$ in $O(kn)$ time by using the standard dynamic programming algorithm for the subset sum problem (see [10]). Therefore we can solve the problem in $O(m + 4^k kn)$ expected time when G contains such a vertex cover and, using $(n, 2k)$-universal sets for derandomization, $O(m + 4^k (2k)^{O(\log k)} kn \log n)$ worst-case time, which is $O(m + n \log n)$ for each fixed k.

Remark 3. This example illustrates that random separation is also useful for finding a solution in a general graph if there is a required solution such that the total number of elements in the solution and its neighbourhood is bounded by a function of k. In the full paper, we will solve several problems with such a property, in particular, some fixed-cardinality optimization problems studied in [4].

3 Extended Random Separation

We have seen in the previous section that random separation is quite useful for solving fixed-parameter problems. In this section, we will extend our basic method to solve a large class of fixed-cardinality optimization problems on degree-bounded graphs.

A random partition (V_g, V_r) is *i-separating*, $i \geq 1$, if there is a solution S such that all vertices in S are green and all other vertices at most distance i away from S are red. We note that our basic idea in random separation is to use a 1-separating partition to separate a solution.

3.1 Maximum Dominating k-Sets for Degree-Bounded Graphs

Let us consider the problem of finding k vertices V' in a graph G of maximum degree d to dominate the maximum number of vertices, i.e., to maximize $|N_G(V')|$. First we note that the dynamic programming approach based on a 1-separating partition does not work as a red vertex can be simultaneously dominated by vertices in several green components. In fact, it seems that no i-separating partition, $i \geq 1$, enables us to use dynamic programming directly based on the information of each green component.

To solve the problem, we use 2-separating partitions together with the new idea of merging green components into clusters. Let (V_g, V_r) be a 2-separating partition of G, and V' a maximum dominating k-set such that all vertices in V' are green and all other vertices that are at most distance 2 away from V' are red. Observe that for any two green components H_1 and H_2, if $d_G(H_1, H_2) \leq 2$ then $V(H_1) \subseteq V'$ iff $V(H_2) \subseteq V'$. Therefore we can merge green components into clusters so that all vertices in a cluster are either all or none in V'.

Let G_H be the graph whose vertices are green components and whose edges correspond to pairs of green components with distance at most 2 in G. Then each connected component of G_H corresponds to a *cluster of green components*, called 2-*cgc*, whose vertices must be either all or none in V'. Furthermore, the distance in G between any two 2-cgcs is at least 3 and thus a red vertex is dominated by at most one 2-cgc. Therefore V' consists of a collection of 2-cgcs.

We can find all 2-cgcs in $O(dn)$ time by using breadth-first search. For each 2-cgc C, we compute the number $\phi(C)$ of vertices dominated by C. Then $\phi(C_1 \cup C_2) = \phi(C_1) + \phi(C_2)$ hold for any two 2-cgcs C_1 and C_2 as no vertex is simultaneously dominated by both C_1 and C_2. Therefore, given a 2-separating partition, we can use the standard dynamic programming algorithm for the (0,1)-knapsack problem based on 2-cgcs to find a maximum dominating k-set in $O(kn)$ time.

Since the probability that a random partition is 2-separating is at least $2^{-k(1+d^2)}$, our algorithm finds, with at least this probability, a maximum dominating k-set in $O((d+k)n)$ time. We can use $(n, k(1+d^2))$-universal sets to derandomize the algorithm so that it runs in $O(n \log n)$ time for fixed k and d.

3.2 Fixed-Cardinality Optimization Problems on Degree-Bounded Graphs

The idea for solving the maximum dominating k-set problem can be generalized to form the base of our *extended random separation* method. We use a random partition (V_g, V_r) of V to obtain, with some probability p, an i-separating partition for some $i \geq 1$ that separates a solution S from the rest of the graph by i layers of red vertices.

Let G_H be the graph whose vertices are green components and whose edges correspond to pairs of green components with distance at most i in G. Then each connected component of G_H corresponds to a *cluster of green components*, called *i-cgc*, whose vertices must be either all or none in S. Furthermore, the distance in G between any two i-cgcs is at least $i+1$.

We merge green components into i-cgcs. Then solution S consists of a collection of i-cgcs. We use dynamic programming based on information of i-cgcs to find an appropriate collection of i-cgcs to produce a solution and derandomize the algorithm by using $(n, t(k,d))$-universal sets for some integer $t(k,d)$.

The extended random separation method is quite powerful and can be used to solve classes of fixed-cardinality optimization problems on degree-bounded graphs that require us to find k vertices (edges) to optimize a value ϕ defined on vertices (edges).[1]

Theorem 4. *Let $k, d \in N$ be fixed constants and $G = (V, E)$ a graph of maximum degree d. Let $\phi : 2^V \to R \cup \{-\infty, +\infty\}$ be an objective function to be optimized. Then it takes $O(n^{\max\{c', c+1\}} \log n)$ time to find k vertices V' in G that optimizes $\phi(V')$ if the following conditions are satisfied:*

1. *For all $V' \subseteq V$ with $|V'| \leq k$, $\phi(V')$ can be computed in $O(g(k,d)n^c)$ time for some function $g(k,d)$ and constant $c > 0$.*
2. *There is a positive integer i computable in $O(h(k,d)n^{c'})$ time for some function $h(k,d)$ and constant $c' > 0$ such that for all $V_1, V_2 \subseteq V$ with $|V_1| + |V_2| \leq k$, if $d_G(V_1, V_2) > i$ then $\phi(V_1 \cup V_2) = \phi(V_1) + \phi(V_2)$.*

[1] We use $\phi(S) = -\infty$ for maximization problems and $\phi(S) = +\infty$ for minimization problems to indicate that S is not a feasible solution.

Proof. (sketch) First we generate a random partition (V_g, V_r) of V. Let V' be an optimal k-solution. Then the probability that (V_g, V_r) is an i-separating partition for V' is at least $2^{-t(k,d)}$ for $t(k,d) = k + kd\sum_{j=1}^{i}(d-1)^{j-1}$.

It takes $O(h(k,d)n^{c'})$ time to compute i, and $O(dn)$ time to find all i-cgcs of an i-separating partition. For each i-cgc C, we delete it if it contains more than k vertices. Otherwise we determine the number of vertices in C and compute $\phi(V(C))$ in $O(g(k,d)n^c)$ time. For any two i-cgcs C_1 and C_2, their distance in G is at least $i+1$ and thus $\phi(V(C_1 \cup C_2)) = \phi(V(C_1)) + \phi(V(C_2))$ when $|V(C_1)|+|V(C_2)| \leq k$. This enables us to use the standard dynamic programming algorithm for 0-1 knapsack problem to find an optimal k-solution in $O(kn)$ time. We can derandomize the algorithm by $(n, t(k,d))$-universal sets to make it run in $O(n^{\max\{c',c+1\}}\log n)$ time. □

Remark 5. Theorem 4 is easily adapted for ϕ being a property of V'. Furthermore, the theorem can be generalized to problems of selecting k disjoint (induced or partial) subgraphs $S_1, S_2, \ldots, S_k$ from a degree-bounded graph (digraph) to optimize the value of an objective function defined on them, provided that the total number of vertices in all S_i's is bounded by a function of k and S_i's are *homogeneous*, for instance, all S_i's are edges, triangles, trees, or planar graphs. Further generalizations to hypergraphs are also possible, and we will discuss these issues in the full paper.

4 Combining with Colour-Coding

As demonstrated in the previous two sections, random separation is very effective in solving fixed-parameter problems on degree-bounded graphs. However, it is much more difficult to solve fixed-parameter problems on graphs of bounded degeneracy. In fact, we can show that several problems that are fixed-parameter tractable for degree-bounded graphs, including the (induced) subgraph isomorphism problem, are W[1]-hard even for 2-degenerate graphs [5].

In this section, we will combine random separation with colour-coding to solve some induced subgraph isomorphism problems for d-degenerate graphs. Let G be a d-degenerate graph whose edges have been oriented so that the outdegree of each vertex is at most d. To solve a fixed-parameter problem for G, we use random separation first to separate a k-solution from its out-neighbours and then use colour-coding for the green subgraph to locate a solution. In other words, we use $k+1$ colours, instead of k colours in colour coding, to form a base of our randomized algorithm, where the role of the extra colour is the same as red in random separation. Surprisingly, this extra colour allows us to solve problems that seem not manageable by colour-coding or random separation alone. We can use universal sets and perfect hash functions together to derandomize the algorithm.

The following simple observation is one of the main reasons that our combined approach works for finding certain isomorphic induced subgraphs in d-degenerate graphs.

Lemma 1. *Let $\vec{G}$ be an arbitrary orientation of a graph G and H a subgraph of G. If $N^+_{\vec{G}}(v) \cap V(H) \subseteq N_H(v)$ for every vertex v in H, then H is an induced subgraph of G.*

Proof. Let $\vec{uv}$ be an edge in $\vec{G}$ between two arbitrary vertices u and v in H. Then $v \in N^+_{\vec{G}}(u) \cap V(H)$ and thus $v \in N_H(u)$. Therefore uv is an edge of H and hence H is an induced subgraph of G. □

We start with the isomorphic induced subtree problem.

Theorem 6. *Let k and d be fixed constants, T a tree on k vertices, and $G = (V, E)$ a d-degenerate graph that contains an induced T-subgraph. Then we can find an induced T-subgraph in G in $O(n)$ expected time and $O(n \log^2 n)$ worst-case time.*

Proof. Arbitrarily choose a leaf of T as the root and define a post-order traversal of T. For convenience, we assume that vertices of T are $1, 2, \cdots, k$ following the post-order traversal. For each vertex i, let T_i denote the subtree of T rooted at i, and $p(i)$ the parent of i in T. Then $i < p(i)$ and k is the root of T.

We orient edges of G so that the outdegree of each vertex is at most d, which is easily done in $O(dn)$ time. For a $(k+1)$-colouring $c : V \to \{0, 1, 2, \ldots, k\}$ of G, an induced T-subgraph T' in G is "well-coloured" if the vertex in T' corresponding to vertex i in T has colour i and every vertex in the out-neighbourhood $N^+_G(T')$ of T' has colour 0.

Given a $(k+1)$-colouring c of V, the following algorithm finds a well-coloured induced T-subgraph T' in G if such a T' exists. To do so, we process vertices v of colour i for each i from 1 to k: roughly speaking, when there is a T_i-subgraph rooted at v, we mark v and add appropriate edges to E'.

Algorithm Iso-Tree

Step 1. Generate a $(k+1)$-colouring $c : V \to \{0, 1, 2, \ldots, k\}$ of G as follows: produce a random partition (V_g, V_r) of V, colour all red vertices V_r by colour 0 and then randomly colour all green vertices V_g by colours in $\{1, 2, \ldots, k\}$.

Step 2. For each vertex v of colour 1, mark it if at most one vertex in $N^+_G(v)$ has colour $p(1)$, and all other vertices in $N^+_G(v)$ have colour 0.

Step 3. For each i from 2 to k, process vertices of colour i as follows. For each vertex v of colour i, mark it if the following conditions are satisfied:
1. For each child j of vertex i in T, there is a marked vertex $u_j \in N_G(v)$ of colour j.
2. If $i \neq k$ then at most one vertex in $N^+_G(v)$ has colour $p(i)$.
3. All other vertices in $N^+_G(v)$ have colour 0.

Add edge vu_j to E' when v is marked.

Step 4. If there is a marked vertex v of colour k, then E' contains the edges of an induced T-subgraph.

The correctness of the algorithm can be established by Lemma 1 and induction on i. For the running time, it is easy to see that the algorithm takes $O(dn)$ time for a given $k+1$ colouring. The probability that an induced T-subgraph in G is well-coloured is at least $2^{-k(d+1)}k^{-k}$, and thus the expected time of the algorithm is $O(2^{k(d+1)}k^k dn)$. To derandomize the algorithm, we use $(n, k(d+1))$-universal sets and then (n, k)-perfect hash functions. Therefore the derandomized algorithm runs in $O(n \log^2 n)$ time for fixed k and d. □

A slight modification to **Iso-Tree** enables us to extend the above theorem to T being an arbitrary k-vertex forest. We can also use the ideas in **Iso-Tree** to find induced k-cycles in graphs of bounded degeneracy.

Theorem 7. *For fixed constants k and d, we can find an induced k-cycle, if it exists, in a d-degenerate graph G in $O(n^2)$ expected time and $O(n^2 \log^2 n)$ worst-case time.*

Proof. (sketch) Basically, we use algorithm **Iso-Tree**. The idea is to use it to find well-coloured induced k-paths with the following modification: mark a vertex v of colour 1 if there are at most two vertices $x, y \in N_G^+(v)$ with $c(x) = 2$ and $c(y) = k$ and all other vertices in $N_G^+(v)$ have colour 0.

Then for each marked vertex v of colour 1, independently do a round of the marking process to find marked vertices $M_k(v)$ of colour k. If there is an edge between v and some vertex in $M_k(v)$, then we find an induced k-cycle in G.

Since we need to do $O(n)$ rounds of independent marking, the algorithm takes $O(n^2)$ expected time and thus $O(n^2 \log^2 n)$ worst-case time after derandomization. □

Note that it is W[1]-hard to find an induced k-path (k-cycle) in a general graph [5].

Remark 8. We can easily extend Theorem 6 and Theorem 7 to deal with weighted graphs. We can also use the idea in **Iso-Tree** to find in $O(n \log^2 n)$ time an induced k-vertex tree (forest) in a d-degenerate graph, which is W[1]-hard for general graphs [5]. Furthermore, it seems possible that we can generalize the idea to find, for any k-vertex graph H of tree-width w, an induced H-subgraph in a d-degenerate graph in $O(n^w \log^2 n)$ time.

5 Concluding Remarks

We have introduced the innovative random separation method for solving fixed-parameter problems and have demonstrated its power through a wide range of such problems. It is quite surprising that this new method is much more powerful and versatile than expected, and we believe that it is a promising and effective tool for solving fixed-parameter problems. For further development, we list a few open problems for the reader to ponder.

1. We feel that the power of random separation has not been fully explored for graphs of bounded degeneracy. What kind of fixed-cardinality optimization problems can be solved for such graphs? Is it possible to obtain some general results, such as those for degree-bounded graphs, for planar graphs?

2. For degree-bounded graphs G, is random separation useful for problems that do not have the properties in Theorem 4? For instance, the problem of deleting k vertices from G to create as many components as possible.
3. For the complexity of our deterministic algorithms, is it possible to remove the $\log n$ factor? For the combined method, is there a direct way to derandomize the algorithms to reduce $\log^2 n$ to $\log n$?
4. Unless P = NP, it is unavoidable that functions $f(k, d)$ in the running times of most of our uniformly polynomial-time algorithms are exponential in both k and d. However, is it possible to reduce $f(k, d)$ to $c_1^k + c_2^d$ for some constants c_1 and c_2 independent of k and d?
5. Is there an $O(n)$ expected time algorithm for finding an induced k-cycle in a d-degenerate graph?
6. Finally, it will be interesting to see how fast our algorithms can run in practice. For implementation purpose, it is useful to fine tune the probability that a vertex is coloured green, and a preliminary test of our randomized algorithm for the exact vertex cover problem shows very encouraging results.

References

1. N. Alon, R. Yuster, and U. Zwick, Color-coding, *J. ACM* 42(4): 844-856, 1995.
2. S. Arnborg, J. Lagergren and D. Seese, Easy problems for tree-decomposable graphs, *J. Algorithms* 12: 308-340, 1991.
3. H.L. Bodlaender, A linear time algorithm for finding tree-decompositions of small treewidth, *SIAM J. Comput.* 25: 1305-1317, 1996.
4. L. Cai, Parameterized complexity of cardinality constrained optimization problems, submitted to a special issue of *The Computer Journal* on parameterized complexity, 2006.
5. L. Cai, S.M. Chan and S.O. Chan, research notes, 2006.
6. J. Chen, I. Kanj, and W. Jia, Vertex cover: further observations and further improvements, *J. Algorithms* 41: 280-301, 2001.
7. B. Courcelle, The monadic second-order logic of graphs I: recognisable sets of finite graphs, *Inform. Comput.* 85(1): 12-75, 1990.
8. R.G. Downey and M.R. Fellows, *Parameterized Complexity*, Springer-Verlag, 1999.
9. M. Frick and M. Grohe, Deciding first-order properties of locally tree-decomposable structures, *J. of the ACM* 48(6): 1184-1206, 2001.
10. J. Kleinberg and E. Tardos, *Algorithm Design*, Pearson, 2005.
11. M. Naor, L.J. Schulman, and A. Srinivasan, Splitters and near-optimal derandomization, *Proc. 36th Annual Symp Foundations of Computer Science*, pp. 182-191, 1995.
12. N. Robertson and P.D. Seymour, Graph minors XIII: the disjoint paths problem, *J. Comb. Ther.(B)* 63(1): 65-110, 1995.
13. J.P. Schmidt and A. Siegel, The spatial complexity of oblivious k-probe hash functions, *SIAM J. Comp.* 19: 775-786, 1990.
14. D. Seese, Linear time computable problems and first-order descriptions, *Mathematical Structures in Computer Science* 6(6): 505-526, 1996.